Working with DbContext

Create and use DbContext object

```
using (var context = new SchoolContext())
{
    //work with context here
}
```

Create Operation

```
context.Students.Add(newStudentObj);
context.SaveChanges();
// or
context.Entry(newStudentObj).State = EntityState.Added;
context.SaveChanges();
```

Update Operation

```
context.Entry(updatedObj).State = EntityState.Modified;
context.SaveChanges();
```

Delete Operation

```
context.Entry(entityObj).State = EntityState.Deleted;
context.SaveChanges();
```

Get the current State of an entity

```
var state = context.Entry<Student>(studentObj).State;
```

Execute native SQL query for entity types

```
var sList = context.Students.SqlQuery("Select * from Student").ToList<Student>();
```

Execute native SQL query for non-entity types

```
string name = contex.Database.SqlQuery<string>("Select name  from Student ").FirstOrDefault<string>();
```

Execute native SQL commands

```
int noOfRowsAffected = context.Database.ExecuteSqlCommand("CUD command");
```

Explicitly load navigation /reference entity

```
context.Entry(student).Reference(s => s.Standard).Load();
```

Explicitly Load Collection

```
context.Entry(student).Collection(s => s.Courses).Load();
```

Find by PrimaryKey

```
var student = context.Students.Find(1);
```

Disable automatic detect changes

```
context.Configuration.AutoDetectChangesEnabled = false
```

Disable lazy loading of navigation properties

```
context.Configuration.LazyLoadingEnabled = false;
```

Disable proxy creation of entity

```
context.Configuration.ProxyCreationEnabled = false;
```

Disable entity validation

```
context.Configuration.ValidateOnSaveEnabled = false;
```

Default Code-First Conventions

Table Name

<Entity Class Name> + ‘s’

Primary key Name

1) Id
2) <Entity Class Name> + ‘Id’ (case insensitive)

e.g. Id or StudentId property becomes primary key of Student by default

Foreign key property Name

By default EF will look for foreign key property with the same name as principal entity primary key name.

If foreign key property does not exists then EF will create FK column in Db table with
<Dependent Navigation Property Name> + ‘_’ + <Principal Entity Primary Key Property Name>

e.g. EF will create Standard_StandardId foreign key column into Students table if Student entity does not contain foreignkey property for Standard where Standard contains StandardId

Null column

EF creates null column for all reference type properties and nullable primitive properties

NotNull Column

EF creates NotNull columns for PrimaryKey properties and non-nullable value type properties

Columns order

Move Primary Key properties to appear first in DB table

Properties mapping to DB

By default all properties will map to database. Use [NotMapped] attribute to exclude property or class from DB mapping

Cascade delete for One-to-Many

Enabled by default

DataAnnotations Attributes

[Table(“name”)],
[Table(“name”, schema=”name”)]

Specifies the database table name and schema (optional) that a class is mapped to.

[key]

Specifies primary key property

[key]
[column(Order=1)]

Specifies that a property is one of the key in composite primary key

[column(“name”,order=)]
[column(“name”,TypeName=)]

Specifies a database column name, order and data type.

[timestamp]

Specifies data type of a column as a row version and makes it concurrency column. This column will be included in where clause of all updates command. It can be applied to only one property of entity class

[ConcurrencyCheck]

Specifies that a property participates in optimistic concurrency checks. It can be applied to multiple properties of entity class.

[Required]

Creates NOTNULL column for a property

[MaxLength(number,Named Parameters)]
[MinLength(number, Named Parameters)]

Specifies maximum and minimum Length of a string property

[StringLength(number, Named Parameters)]

Specifies the size of a string property

[ForeignKey(“name”)]

Denotes a property used as a foreign key in a relationship. The annotation may be placed on the foreign key property and specify the associated navigation property name, or placed on a navigation property and specify the associated foreign key name.

[NotMapped]

Skips DB column creation for a property

[InverseProperty(“name”)]

Specifies the inverse of a navigation property that represents the other end of the same relationship.

[ComplexType]

Specifies complex type.

DataAnnotations Attributes

[DatabaseGenerated(DatabaseGeneratedOption.Computed)]

Sets DB column as computed

[DatabaseGenerated(DatabaseGeneratedOption.Indentity)]

Sets DB column as identity

[DatabaseGenerated(DatabaseGeneratedOption.None)]

Sets DB column as none

Fluent API – Configure Entity Type

Entity to Table mapping

```
modelBuilder.Entity<Student>().ToTable("StudentInfo");

modelBuilder.Entity<Student>()
            .ToTable("StudentInfo", "dbo");
```

Entity splitting into two tables

```
modelBuilder.Entity<Student>()
            .Map(s => { s.Properties(p => new { p.Id,
                                                p.Name });
                        s.ToTable("Student");
             })
             .Map(s => { s.Properties(p => new {
                                 p.Address, p.City});
                         s.ToTable("StudentAddress");
                       });
```

Two entities map to one table

```
modelBuilder.Entity<Student>().ToTable("Student");

modelBuilder.Entity<StudentAddress>()
            .ToTable("Student");
```

Configure complex type

```
modelBuilder.ComplexType<Student>();
```

Entity should not mapped with DB table

```
modelBuilder.Ignore<Student>();
```

Fluent API – Property Configuration

Set key property

```
modelBuilder.Entity<Student>().HasKey(s =>
                                    s.StudentID);
```

Set composite key properties

```
modelBuilder.Entity<Student>()
            .HasKey(s => new { s.StudentID,
                               s.StudentSubID });
```

Set column order

```
modelBuilder.Entity<Student>()
            .Property(s =>  s.StudentSubID)
            .HasColumnOrder(5);
```

Set DB column datatype

```
modelBuilder.Entity<Student>()
            .Property(s => s.StudentName)
            .HasColumnType("varchar");
```

Set Column name

```
modelBuilder.Entity<Student>()
            .Property(s => s.StudentName)
            .HasColumnName("Name");
```

Set parameter name in StoredProcedure

```
modelBuilder.Entity<Student>()
            .Property(s => s.StudentID)
            .HasParameterName("Id");
```

Set Null column

```
modelBuilder.Entity<Student>()
            .Property(s =>  s.StudentName)
            .IsOptional();
```

Set NotNull column

```
modelBuilder.Entity<Student>()
            .Property(s => s.StudentName)
            .IsRequired();
```

Fluent API –Property Configurations

Set MaxLength for string column

```
modelBuilder.Entity<Student>()
            .Property(s => s.StudentName)
            .HasMaxLength(50);
```

Set fixed length property (char)

```
modelBuilder.Entity<Student>()
            .Property(s => s.StudentName)
            .HasMaxLength(50);

modelBuilder.Entity<Student>()
            .Property(s => s.StudentName)
            .IsFixedLength();
```

Set Unicode string column

```
modelBuilder.Entity<Student>()
            .Property(s =>    s.StudentName)
            .IsUnicode();
//or
modelBuilder.Entity<Student>()
            .Property(s => s.StudentName)
            .IsUnicode(false);
```

Set decimal column precision

```
modelBuilder.Entity<Student>()
            .Property(s =>  s.Height)
            .HasPrecision(9, 4);
```

Set concurrency property:

```
modelBuilder.Entity<Student>()
            .Property(p => p.StudentName)
            .IsConcurrencyToken();
//or
modelBuilder.Entity<Student>()
            .Property(p => p.RowVersion)
            .IsRowVersion();
```

Set Identity Column

```
modelBuilder.Entity<Student>()
            .Property(p => p.StudentID)
            .HasDatabaseGeneratedOption(
               DatabaseGeneratedOption.Identity);
```

Fluent API – Relationship Configurations

One-to-zero-or-one

```
modelBuilder.Entity<StudentAddress>()
            .HasKey(e => e.StudentId);

modelBuilder.Entity<Student>()
            .HasOptional(s => s.Address)
            .WithRequired(ad => ad.Student)
```

One-to-One

```
modelBuilder.Entity<StudentAddress>()
            .HasKey(e => e.StudentId);

modelBuilder.Entity<Student>()
            .HasRequired(s => s.Address)
            .WithRequiredPrincipal(ad =>
                                ad.Student);
```

One-to-Many

```
modelBuilder.Entity<Student>()
            .HasRequired<Standard>(s =>
                                s.Standard)
            .WithMany(s => s.Students);
```

Many-to-Many

```
modelBuilder.Entity<Student>()
            .HasMany<Course>(s => s.Courses)
            .WithMany(c => c.Students)
            .Map(cs => {
              cs.MapLeftKey("StudentRefId");
              cs.MapRightKey("CourseRefId");
              cs.ToTable("StudentCourse");
        });
```

Property DataType	Mapped DB Column DataType	Mapped Key column DataType
Int	int	int, Identity column increment by 1
string	nvarchar(Max)	nvarchar(128)
decimal	decimal(18,2)	decimal(18,2)
float	real	real
byte[]	varbinary(Max)	varbinary(128)
datetime	datetime	datetime
bool	bit	bit
Byte	tinyint	tinyint
short	smallint	smallint
long	bigint	bigint
double	float	float
char	No mapping	No mapping
sbyte	No mapping (throws exception)	No mapping
object	No mapping	No mapping

Database Configurations

Set DB Initializer

```
Database.SetInitializer<Context>(new
            DropCreateDatabaseAlways<Context>());

Database.SetInitializer<Context>(new
    DropCreateDatabaseIfModelChanges<Context>());

Database.SetInitializer<Context>(new
           CreateDatabaseIfNotExists<Context>());
```

Remove Plural Table Name Convention

```
Using
System.Data.Entity.ModelConfiguration.Conventions;

modelBuilder.Conventions
       .Remove<PluralizingTableNameConvention>();
```